IPS **blok 4, 2022**

IPS assignment 5

7. juni 2022

Anders Persson (tqc110)

Task 1

I follow the example from the LABS and group rejecting and accepting into 2 groups

$$G_1 = \{1, 2, 4, 5, 6, 7, 8\} = REJECT$$

$$G_2 = \{3\} ACCEPT$$

G_2 is a singleton group so it is marked as consistent and we still have one unmarked group, i will now check its consistency by making a table of the transitions.

G_1	0	1
1	G_1	G_1
2	G_1	G_2
4	G_2	G_1
5	G_1	G_1
6	G_2	G_1
7	G_1	G_1
8	G_1	G_2

Looking at the table i notice that G_1 is non-consistent, and i split it into subgroups and erase the marks. I now have the follwoing groups

$$G_2 = \{3\}$$

$$G_3 = \{1, 5, 7\}$$

$$G_4 = \{2, 8\}$$

$$G_5 = \{4, 6\}$$

I now choose G_3 and check its consistency.

G_3	0	1
1	G_4	G_5
5	G_4	G_5
7	G_3	G_3

This one is still inconsistent so i split it into new sub groups G_6 and G_7.

$$G_6 = \{1, 5\}$$

$$G_7 = \{7\}$$

I now look at G_4 and see that it is consistent.

G_4	0	1
2	G_7	G_2
8	G_7	G_2

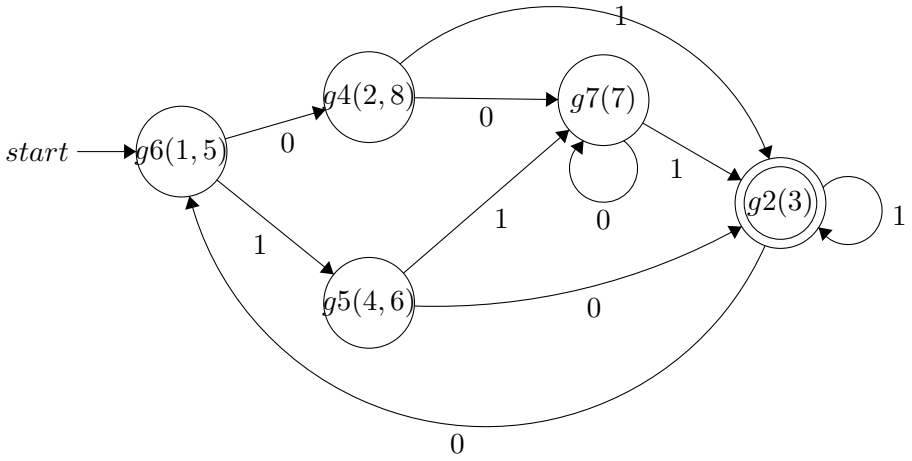
We then take a look at G_5 and this is also consistent.

G_5	0	1
4	G_2	G_7
6	G_2	G_7

The only group left to check that is unmarked is G_6 as G_7 is a singleton and thus it is consistent.

G_6	0	1
1	G_4	G_5
5	G_4	G_5

We now see that all the groups are consistent and thus we can form a minimal DFA and the result is drawn underneath:



Task 2

a)

(i)

This regular expressions works by first using the Boolean operator "or"and either matching with the number 5 as this is the lowest number divisible with 5 that doesn't start with a 0. And if it doesn't match then it checks for a number that starts with any number that's not 0, and from there all numbers are fair game as long as the number ends on 0 or 5 so it is divisible with 5.

```
5|[1-9][0-9]*(0|5)
```

(ii)

This regular expression works by matching on excluding 5 from the number 0 or more times and then matching a 5 3 times so we get exactly 3 5s as the object was. So it can be any number other than 5 or 5 and this happens 3 times to make sure we don't match 4 5s for example by ending on the exclusion of a 5.

```
[^5]*5[^5]*5[^5]*5[^5]*
```

(iii)

This regular expression works by first matching the first group between 0 and 1 time using the ? operator. Inside this group i am excluding 0 and 5 from 0 to unlimited times because if i didn't exclude 5 from here ts regular expression would be valid for a number like 5555 because 5 wasn't excluded. 0 is excluded because if a number has to be divisible by 5 it can't start on a 0. Next is just one of the 3 5s that is needed followed by any number as many times as needed excluding 5. And once more. and then the last 5 of the 3 5s we needed. then lastly we can have any number excluding 5 because we already have 3 5s. And then lastly making sure it ends on a 0 if it doesn't end on one of the 3 5s making sure it is still divisible by 5.

```
([^50]*)?5[^5]*5[^5]*5([^5]*0)?
```

b)

(i)

The first language i consider to be non regular or irregular as there is infinitely many possibilities thus it is infinite. And because one of the requirements for a regular language is that it can be accepted by NFA and DFA and with infinitely many states that is not possible

(ii)

The next language however is considered regular as it bounded by 1 million and because you technically could write down all the possibilities even though it would take quite a long time it is considered finite and NFA and DFA can count all the states.

Task 3

After reading section 2.3 in the book and reading the lecture slides again i have come up with this unambiguous grammar. The higher in the syntax tree you could make from this the lower the precedence.

```
E  ->  E || T | T
T  ->  R && T | R
R  ->  (E) | "true" | "false"
```

Task 4

a)

I eliminate left recursion.

```
E  -> num E'
E' -> E + E'
E' -> E * E'
E' ->
```

b)

I do left factorization of the grammar obtained from question a).

```
E   -> num E'
E'  -> E Aux
E'  ->
Aux -> +E'
Aux -> *E'
```

c)

I compute the Nullable and FIRST sets for every production and the FOLLOW sets for every nonterminal on the grammar obtained from question b).

	$Nullable$	$FIRST$
$E \rightarrow \mathbf{num}E'$	$false$	**num**
$E' \rightarrow EAux$	$false$	**num**
$E' \rightarrow$	$true$	
$Aux \rightarrow +E'$	$false$	$+$
$Aux \rightarrow *E'$	$false$	$*$

	$FOLLOW$
E	$+, *, \$$
E'	$+, *, \$$
Aux	$+, *, \$$

d)

Didn't manage to understand how to make the Look-Ahead table and Recursive-Descent Parser. But the LL(1) parse table was pretty straightforward after doing the Nullable and FIRST sets.

	num	$+$	$*$	\$
E	$E \rightarrow \mathbf{num}E'$			
E'	$E' \rightarrow EAux$	$E' \rightarrow$	$E' \rightarrow$	$E' \rightarrow$
Aux		$Aux \rightarrow +E'$	$Aux \rightarrow *E'$	

Task 5

Im doing Task 5 to hopefully make up for not doing the entirety of task 4.

a)

I add a new start production and compute FOLLOW(T).

I add the production $T' \rightarrow T$. As well as the production $T'' \rightarrow T'\$$ so i can compute the FOLLOW.

T''	$\rightarrow T'\$$	$\$ \in FOLLOW(T')$
T'	$\rightarrow T$	$FOLLOW(T') \subseteq FOLLOW(T)$
T	$\rightarrow T => T$	$=> \in FOLLOW(T)$
T	$\rightarrow T * T$	$* \in FOLLOW(T)$
T	$\rightarrow$ **int**	

Which then evaluates to:

$$FOLLOW(T') = \{\$\}$$
$$FOLLOW(T) = \{\$, =>, *\}$$

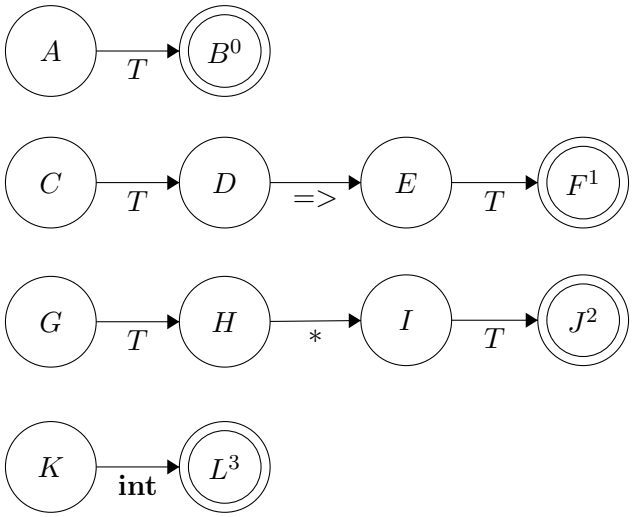
b)

I will now construct an SLR parser table for the grammar. And to do that i need to do a few things first. I follow the guide on Constructing SLR Parse tables from the book and lecture slides.

I start by numbering the productions of the grammar.

$$\begin{array}{lllll} 0: & T' & \rightarrow & T \\ 1: & T & \rightarrow & T => T \\ 2: & T & \rightarrow & T * T \\ 3: & T & \rightarrow & \textbf{int} \end{array}$$

Then i need to make NFAs for each of the productions. (the number of the production is seen in the accept state)



I add the ε-transitions. Every time where a nonterminal transition N is possible an ε-transition is inserted pointing to the initial states of the NFAs corresponding to N as explained in the lecture slides (i drew the actual NFA with ε-transitions, but since there were 15 different ε-transitions it quickly became hard to keep track of them all so instead i kept it nice and simple with this table underneath):

	ε
A	C, G, K
C	C, G, K
E	C, G, K
G	C, G, K
I	C, G, K

And then i convert the NFA to a DFA using a table as seen below:

$State$	NFA	$=>$	$*$	**int**	T
0	A, C, G, K			$s2$	$g1$
1	B, D, H	$s3$	$s4$		
2	L				
3	E, C, G, K			$s2$	$g5$
4	I, C, G, K			$s2$	$g6$
5	F, D, H	$s3$	$s4$		
6	J, D, H	$s3$	$s4$		

c)

We then need to evaluate the accept/reduce actions according to the FOLLOW sets:

I use the rules from the assignment question they are:

1. (i): Operator * binds tighter than =>
2. (ii): Operator * is left associative, and => is right associative

Using those rules i end up with the following table, and the only rows where i had conflicts are 5 and 6. Look in the table below for how i resolved these (Changes from the first table to this one are marked with bold).

State	*NFA*	=>	$*$	**int**	$	T
0	A, C, G, K			$s2$		$g1$
1	B, D, H	$s3$	$s4$		**acc**	
2	L	**r3**	**r3**		**r3**	
3	E, C, G, K			$s2$		$g5$
4	I, C, G, K			$s2$		$g6$
5	F, D, H	$s3$	$s4$		**r1**	
6	J, D, H	**r2**	**r2**		**r2**	

So to explain the conflicts i had a conflict in state 5 where => is shifting on => or reducing to production 1. But i remember the rules from the assignment question and since => is right associative a shift is still performed.

There was also a conflict with * in state 5. Again between shifting on * or reducing to production 1. But according to the rules * binds tighter thus the shift is still performed.

Then there were two conflicts in state 6 on => and * again between shifting or reducing to production 2. * binds tighter and * is left associative thus we reduce both to production 2.